Computer Science 271
Project 1000
Due Friday, April 28

---

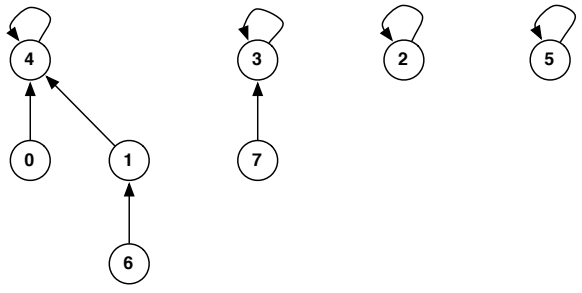
*You will complete this project with a **new partner**. Both individuals are expected to contribute equally to all parts of the project.*

1. Implement a disjoint set forest, with union-by-rank and path compression, following section 21.3 in your textbook. To get you started, here is a skeleton of the header file:

```
template <class T>
class DSNode
{
   // you write this
};

template <class T>
class DisjointSets
{
   public:
      DisjointSets();                                // default constructor
      DisjointSets(int size);                        // constructor with given capacity
      DisjointSets(const DisjointSets<T>& ds);       // copy constructor
      ~DisjointSets();                               // destructor

      DSNode<T>* makeSet(T* x);                      // make a new singleton set containing data x
      void unionSets(DSNode<T>* x, DSNode<T>* y);    // union the disjoint sets containing data x and y
      DSNode<T>* findSet(DSNode<T>* x);              // return the representative of the set containing x

      DisjointSets<T>& operator=(const DisjointSets<T>& ds);  // assignment operator

      std::string toString();            // return a string representation of the disjoint set forest

   private:
      void link(DSNode<T>* x, DSNode<T>* y);

      DSNode<T> **elements;   // array of nodes in the forest
      int capacity;           // size of elements array
      int length;             // number of elements in the forest
};

class FullError { };      // full exception
class NotFoundError { };  // element not found exception
```

   To match the pseudocode in your textbook (and the time complexity), `makeSet` returns a pointer to a node in the disjoint set forest. Then the other two methods take node pointers as parameters.

   The `toString` method should return a string representing the disjoint set forest by iterating over an array of nodes and printing ranks and parent relationships. For example, if the disjoint set forest looked like the following



then your `toString` method should return the following string

```
0:0 -> 4:2
1:1 -> 4:2
2:0
3:1
4:2
5:0
6:0 -> 1:1 -> 4:2
7:0 -> 3:1
```

where each `i:j` represents a data element `i` and its rank `j`. As in previous assignments, we assume that type `T` has the stream insertion operator defined for it. (In this example, I am assuming that the key value for each data element happens to be the index at which the node for that element is stored in the array. Otherwise, the elements won't necessary print in this order.)

As usual, include pre/postconditions, throw exceptions where appropriate, and include unit tests.

2. Write a vertex class and a weighted, directed graph class. (The class can also represent an undirected graph if, for every edge (u, v) there is also an edge (v, u).) Your graph class should be based on an adjacency list representation but may also contain a matrix of edge weights. Each vertex should have a unique integer identifier between 0 and $n - 1$. This identifier will be used as the index of the vertex in the adjacency list. The vertex class also needs to contain any attributes used by algorithms implemented in your graph class (below).

   A constructor for your graph class should read a graph in from a file. The file will contain the number of vertices in the graph followed by an adjacency matrix representation of the weighted graph. For example, the following file represents a particular weighted, undirected graph with 5 vertices:

   ```
   5
   0 1 2 0 0
   1 0 1 1 0
   2 1 0 0 3
   0 1 0 0 1
   0 0 3 1 0
   ```

   (a) Include a method

   ```
   void Graph::dfs()
   ```

   that performs a depth-first search of the graph. Your method should print the vertices in the order they are visited.

   (b) Include a method

   ```
   void Graph::Kruskal()
   ```

   that uses Kruskal's algorithm to construct a minimum spanning tree of an undirected weighted graph. The method should simply print the edges in the MST. You will need to use several of your previous template classes to accomplish this. At a minimum, you must use your disjoint sets forest, but you will probably also want to use your list and minimum priority queue classes. Think carefully about this; the pseudocode is quite simple but it will require some care to implement properly.

   As usual, include pre/postconditions, throw exceptions where appropriate, and include unit tests. Also, remember to implement the copy constructor, destructor, and assignment operator.

Submit your individual source files and a single PDF containing your code, following the instructions in the previous projects. Just submit one submission per group. Be sure to indicate the names of all group members on all of your submitted files.